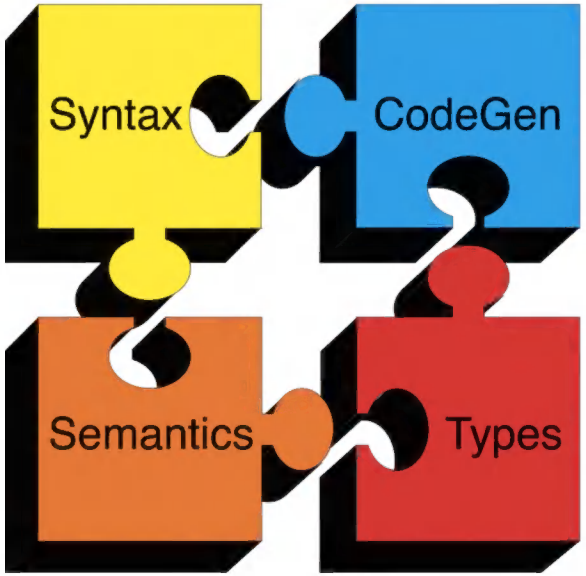


Compilers

Mark and Sweep

Mark and Sweep

- When memory runs out, GC executes two phases
  - the mark phase: traces reachable objects
  - the sweep phase: collects garbage objects

- Every object has an extra bit: the mark bit
  - reserved for memory management
  - initially the mark bit is 0
  - set to 1 for the reachable objects in the mark phase

Mark and Sweep

```
// mark phase

let todo = { all roots }
while todo ≠ ∅ do
    pick v ∈ todo
    todo ← todo - { v }
    if mark(v) = 0 then      // v is unmarked yet
        mark(v) ← 1
        let v1,...,vn be the pointers contained in v
        todo ← todo ∪ {v1,...,vn}
    fi
od
```

Mark and Sweep

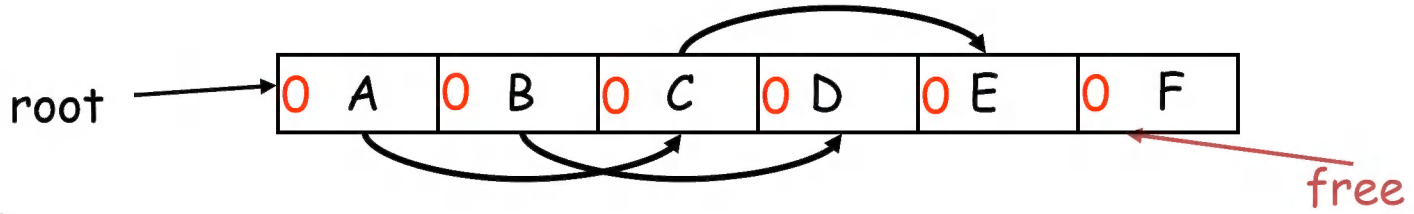
- The sweep phase scans the heap looking for objects with mark bit 0
  - these objects were not visited in the mark phase
  - they are garbage
- Any such object is added to the free list
- Objects with a mark bit 1 have their mark bit reset to 0

Mark and Sweep

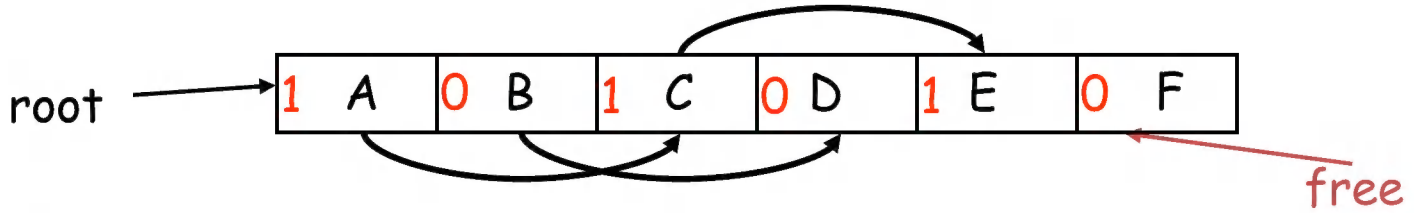
```
// sweep phase
// sizeof(p) is the size of block starting at p

p ← bottom of heap
while p < top of heap do
  if mark(p) = 1 then
     mark(p) ← 0
  else
     add block p...(p+sizeof(p)-1) to freelist
  fi
  p ← p + sizeof(p)
od
```

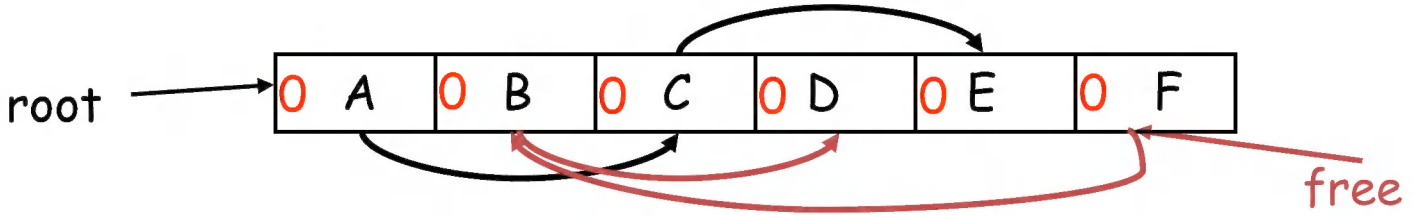
Mark and Sweep



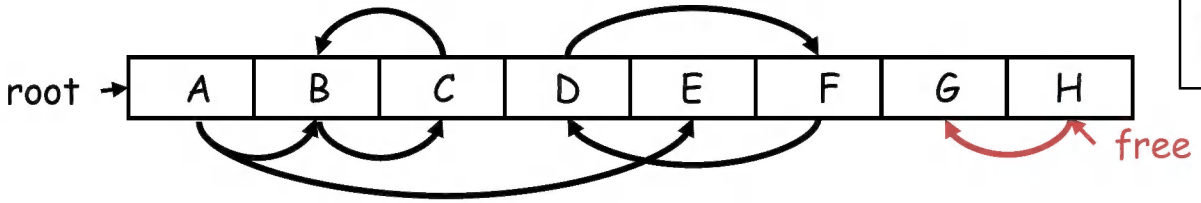
After mark:



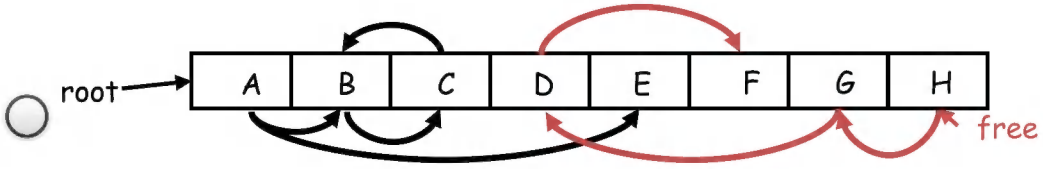
After sweep:

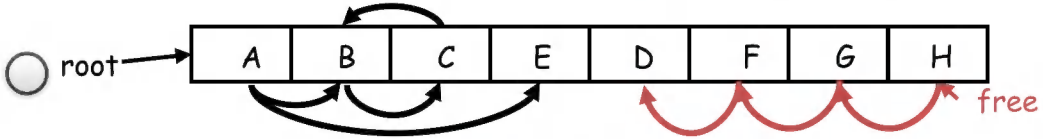


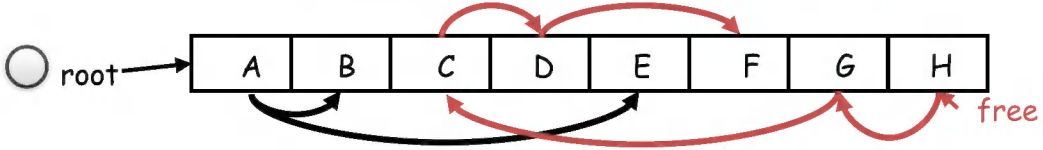
Mark and Sweep

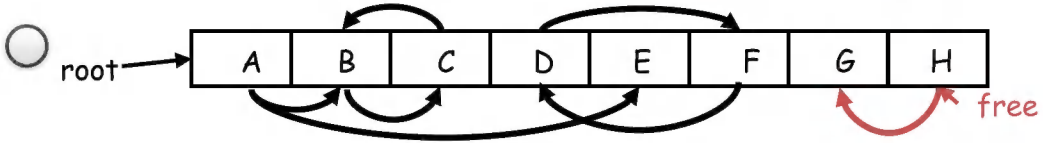


Choose the correct final heap after mark and sweep garbage collection.









Mark and Sweep

- While conceptually simple, this algorithm has a number of tricky details
  - typical of GC algorithms

- A serious problem with the mark phase
  - it is invoked when we are out of space
  - yet it needs space to construct the todo list
  - the size of the todo list is unbounded so we cannot reserve space for it a priori

Mark and Sweep

- The todo list is used as an auxiliary data structure to perform the reachability analysis
- There is a trick that allows the auxiliary data to be stored in the objects themselves
  - pointer reversal: when a pointer is followed it is reversed to point to its parent
- Similarly, the free list is stored in the free objects themselves

Mark and Sweep

Mark and Sweep

- Space for a new object is allocated from the new list
  - a block large enough is picked
  - an area of the necessary size is allocated from it
  - the left-over is put back in the free list
- Mark and sweep can fragment the memory
- Advantage: objects are not moved during GC
  - no need to update the pointers to objects
  - works for languages like C and C++